To appear in the *International Journal of Remote Sensing*
Vol. 00, No. 00, Month 20XX, 1–22

GVLiDAR: An Interactive Web-based Visualization Framework to Support Geospatial Measures on LiDAR Data

David Deibe[a]*, Margarita Amor[a], Ramón Doallo[a], David Miranda[b] and Miguel Cordero[b]

[a]*Facultade de Informática da Coruña, Universidade da Coruña (UDC), A Coruña, Spain*; [b]*Escola Politécnica Superior de Lugo, Universidade de Santiago de Compostela (USC), A Coruña, Spain*

(*Received 00 Month 20XX; accepted 00 Month 20XX*)

In recent years LiDAR (Light Detection And Ranging) technology has experienced a noticeable increase in its relevance and usage in a number of scientific fields. Therefore, software capable of handling LiDAR data becomes a key point in those fields. In this paper, we present GVLiDAR (GPU-based Viewer LiDAR), a novel web framework for visualization and geospatial measurement of LiDAR data point sets. The design of the framework is focused on achieving three key objectives: performance in terms of real-time interaction, functionality and online availability for the LiDAR datasets. All LiDAR files are pre-processed and stored in a lossless data structure which minimizes transfer requirements and offers an on-demand LiDAR data web framework.

Keywords: LiDAR; WebGL; GPU; Geospatial measures; On-demand data

1. Introduction

In the past few decades the usage of remote-sensing techniques has experienced a notable increase, driven by the need to obtain a wide range of Earth surface information while achieving satisfactory cost-efficiency ratios, high data precision and short data acquisition times. This information is used by scientists and government institutions to detect environmental changes, to classify urban and rural landscapes or to conduct efficient public policy planning, along with an extensive list of further applications (Tarolli 2014). Among all remote-sensing techniques, airborne and terrestrial LiDAR (*Light Detection and Ranging*) technologies stand out as some of the most important and useful available nowadays.

LiDAR technology provides very useful high-resolution data in the form of point clouds that can be applied in a wide variety of fields, such as agriculture, archaeology, biology, geology or forestry. Earth-science applications of LiDAR include coastal-change studies (Sallenger Jr. et al. 1999), monitoring of landslides (Ventura et al. 2011), measurement of volcanic deformation (Hasegawa, Sato, and Iwahashi 2007), identification of faults (Arrowsmith and Zielke 2009), land-cover classification (Buján et al. 2013), forest inventory and biomass estimation (Boudreau et al. 2008; González-Ferreiro et al. 2013) or structures on the geometry of fault scarps (Brunori et al. 2013), among others. Furthermore, the combination of LiDAR data with other types of remotely-sensed imagery makes it possible to obtain accurate

*Corresponding author. Email: david.deibe@udc.es mappings of land covers which are very useful in Earth surface monitoring (Zhang 2010; Yan, Shaker, and El-Ashmawy 2015).

Other popular terrain 3D models, such as 2D grid, 3D grid or Triangulated Irregular Network (TIN), which in some cases are derived from LiDAR data, have also been widely used for qualitative and quantitative analysis. 2D grid is a set of sampled points representing measures of altitude or elevation and stored at regular intervals. TIN samples heights irregularly to represent more-detailed areas using more samples. Hybrid models that combine 2 D grid-based models and TIN meshes may add details to specific regions, without increasing the overall grid resolution (Yang, Shi, and Li 2005). Hybrid representation that does not alter the original data and allows the adaptive tessellation of the grid cells in the neighbourhood of the TIN structure was presented in (Bóo, Amor, and Döllner 2007; Paredes et al. 2011). On the other hand, 3D grid represents the terrain as occupied voxel that allows the modelling of various volumetric features; nevertheless, this representation is very slow at rendering large and detailed terrains. In (Koca and Güdükbay 2014), a novel representation, that combines 2D and 3D approaches in order to allow creating terrains with caves, overhangs, cliffs and arches is presented. They provide high horizontal resolution, which may improve the accuracy for recognizing certain topographic features, but fall short in recognizing some structures, such as drainage ditches or levees, which may be misinterpreted during topographic analysis if high-resolution data are not used (Tarolli and Tarboton 2006). While LiDAR models are capable of clearly show buildings, trees or sea ice surface roughness (Landy, Komarov, and Barber 2015), in most of the other digital models those kinds of structures may be barely recognizable or in existent.

All data gathered by LiDAR sensors would be useless without software capable of making use of it. Visualization frameworks bring the possibility of easily exploring and interpreting LiDAR datasets. Nevertheless, the rapid increase in the quantity and density of the LiDAR data obtained demands more efficient and powerful frameworks capable of handling millions of points. With the competition between enterprises for obtaining higher resolutions and acquisition speeds many visualization frameworks not properly updated may become obsolete and unable to handle the point clouds gathered by the newer sensors. Furthermore, it is a key point to give final users the freedom to work on any operating platform without any restriction. Examples of this kind of software can be found in (Kuder and Zalik 2011) and (Lewis, Elhinney, and McCarthy 2012), where a web-based visualization framework for LiDAR data can be found.

In this article we present GVLiDAR, a novel LiDAR framework based on WebGL technology (Parisi 2012; The Khronos Group Inc. 2015b). GVLiDAR is a scalable client-server system for the visualization and interactive manipulation of LiDAR data, with the capability of performing geospatial measurements directly over the 3D point clouds and obtaining data on-demand. The design of the framework is focused on achieving three key objectives: performance in terms of real-time interaction, functionality and online availability for the LiDAR datasets. GVLiDAR can be used on any platform or operating system. The only requirement is a browser with WebGL support. WebGL is based on OpenGL ES 2.0 (OpenGL for Embedded Systems)(Munshi, Ginsburg, and Shreiner 2008) that was developed for handheld devices such as smartphones and tablets.

GVLiDAR development and specially its measurement tools were guided by a group of agroforestry engineers. GVLiDAR offers a variety of possibilities not presented in other similar solutions which was a long time demand from some LiDAR users. Current applications offer a small amount of basic measurement tools and

Table 1. Data contained into one data record of LAS (format 0).

Item	Format	Size	Required
X	long	4 bytes	*
Y	long	4 bytes	*
Z	long	4 bytes	*
Intensity	unsigned short	2 bytes	
Return Number	3 bits (bits 0 - 2)	3 bits	*
Number of Returns	3 bits (bits 3 - 5)	3 bits	*
Scan Direction Flag	1 bit (bit 6)	1 bit	*
Edge of Flight Line	1 bit (bit 7)	1 bit	*
Classification	unsigned char	1 bytes	*
Scan Angle Rank	char	1 bytes	*
User Data	unsigned char	1 bytes	
Point Source ID	unsigned short	2 bytes	*

they are not focused on the needs of professionals that use LiDAR point clouds. This lack of specialization, can be observed, for instance, measuring heights of structures, in many cases, they must be manually calculated by the user from the angles given by ‘angle tools’ or even they cannot be done in some cases. Area measures provided by most applications are only projected horizontally making very hard or impossible to measure facades or slopes. Furthermore, they do not allow to triangulate points in order to accurately measure complex surfaces or volumes. GVLiDAR provides measurement tools for all these situations and its visualization techniques are focused on rendering as much as points as possible in order to support high precision measures.

The article is organized as follows. Some concepts about LiDAR Data Technology and LiDAR frameworks currently available are briefly explained in Section 2. Section 3 describes the internal structure of our GVLiDAR framework. In Section 4 we present different performance tests realized on different platforms. Finally, the main conclusions and future work are presented in Section 5.

2. LiDAR Data Technology

Many LiDAR applications use the LAS file format, a standard in the field of LiDAR solutions. Its specification was created by the (American Society for Photogrammetry and Remote Sensing 2011) (ASPRS). The objective of LAS format is to provide an open format for different LIDAR hardware and software tools.

The original LAS file is intended to contain LIDAR (or other) point cloud data records. The data are stored in this format from software which combines GPS, *Inertial Measurement Unit* (IMU), and laser pulse range data to produce x, y, and z point data.

Specifically, LAS files contain binary data consisting of an initial header block, any number of optional variable length records (VLRs), a block with all the point data records and finally any number of optional extended variable length records (EVLRs). The header block contains generic information about the file, such as the number of point data records stored, the byte position where they begin, or the date when the data were obtained. Both the VLRs and EVLRs are optional blocks that can be used by the different LiDAR software in order to store information which they may require.

Table 1 shows properties of a single point stored in a LAS file; specifically, those which follow the format 0 (LAS specification defines different *Point Data Record* formats, each with different properties).

2.1. *LiDAR frameworks*

In recent years, the use of LiDAR technologies has increased along with the number of software designed to handle this kind of data, both web and desktop applications, specifically designed to work with LiDAR or as part of more general software. Among the frameworks available, we have analysed some of the most commonly used or known.

(1) *LAStools* (3D visor) (Rapidlasso 2015): This is probably the most commonly used desktop application for LiDAR data. It is free, easy to use and includes several processing tools. Its 3D visor performs many different visualization options such as intensity, classification, height, number of return, RGB or scan direction. Users can filter the point cloud by point properties, or create a full 3D model by triangulating the point set. Nevertheless, the 3D visor offers a low performance, rendering over 4,700,000 points under 10 frames per second (FPS). Furthermore, it only works with local files loaded by the users and it lacks geospatial measuring tools over the 3D point cloud.

(2) *Fugro Viewer* (Fugro Geospatial Services 2015): Simple and easy-to-use desktop visor, it provides different visualization options, such as intensity, height, classification, return number or RGB. A complete 3D model can be obtained through a triangulation of the point cloud. There are two visualization modes: 2D and 3D. The 2D mode applies Level of Detail (LOD) techniques over the points set; therefore, when the camera is near, all points are rendered, but from a far point of view only a subset of the points are shown. The 3D mode achieves high performance, although it only renders the visible points that are shown in the 2D window, so that, when zooming over the 3D model, no more points are loaded and detail is lost. The LiDAR data has to be provided by the users. Finally, a tool for measuring distances is available, but only over the 2D image.

(3) *IDECanarias* (Gobierno de Canarias 2015): Web application well known in Spain which provides LiDAR data from the Canary Islands on-demand. It has low retrieval times, around 5 seconds for 350,000 points (the maximum amount of points retrieved), and is easy to use. Nevertheless, in many systems it does not obtain a fluid interaction due to its Flash programming, achieving about 36 FPS with 250,000 points. Areas of a few square meters and large areas of several square kilometres are rendered with almost the same number of points. While the detail obtained in the former case may be good, in the second case it may not be enough for recognizing certain land features or structures. The application has limited visualization options (intensity, height and mixed) and it lacks geospatial measuring tools over the 3D point clouds.

(4) *LiDAR Online* (3D visor) (LiDAR Online 2015): One of the best known web visors is based on *Dielmo 3D* technology (Dielmo 3D S.L 2015). It allows LiDAR data to be obtained on-demand from various locations around the world. The 3D visor has good visualization options (classification, intensity, height and RGB) and offers highly satisfactory performance results, up to 60 FPS displaying around 1 million points. Nevertheless, transferring more than 1 million points were not allowed because, like *IDECanarias*, the number of points shown barely change along with the size of the selected area: small and large areas are retrieved with almost the same number of points, so the last ones are displayed with a lower detail. Furthermore, the times are quite long; more than a minute to retrieve 1 million points, and sometimes the process

may get blocked. It lacks geospatial measuring tools over the 3D point clouds.

(5) *Plas.io* (Uday Verma 2015): This web application is based on WebGL. It focuses on the visualization of local files (in LAS and LAZ formats) achieving 45 FPS, handling around 16 million points. It offers different rendering modes, such as RGB, intensity or height, but lacks geospatial measuring tools over the 3D point clouds.

(6) *Lidarview* (Alexander Krivutsenko 2015): Another web application based on WebGL that offers good performance with around 50 FPS handling 10 million points. It implements some visualization options such as intensity, classification, height and RGB. It is easy to use but only works with local files loaded by the users (LAS or XYZ file formats). The visor is limited to 10 million point clouds and lacks geospatial measuring tools over the 3D point cloud.

(7) *Potree* (Markus Schtz 2015): This web application, also based on WebGL, is an interactive out-of-core rendering solution for massive datasets (over 1 billion points) using a multi-resolution octree. It has the most similar features to our proposal; however, it is conceived as a general-purpose point rendering engine and does not focus solely on LiDAR data. *Potree* allows direct measurement on the 3D images. Nevertheless, its measurement tools are basic and not focused on LiDAR data, it does not allow to get information such as the distance of a point to a plane, the area of complex surfaces or the square meters of facades (required by professionals in the fields of engineering and architecture) to be obtained. Furthermore, it provides no geographic information on the images, such as the reference system used or the geographic coordinates of each point. Therefore, we can consider *Potree* as a tool mainly focused on the fast visualization of large areas of land or highly detailed objects, prioritizing realistic rendering against the functionality of the measurement tools.

In summary, some of these frameworks have a moderate performance in terms of FPS; some of them can only handle small number of LiDAR points, while others can handle a large number of points but with relatively low performance. *Potree* achieves real-time interaction, handling billions of points, but is mainly focused on fast visualization providing basic measurement tools. Some frameworks, such as *IDECanarias*, *LiDAR Online* or *FugroViewer*, apply LOD (Moller, Haines, and Hoffman 2008) algorithms in order to manage large areas and reduce the total number of points. These LOD algorithms transform the original point cloud into a simplified version so they improve their performance, but this simplification may cause some inaccuracy during measurements or identifying land features. Furthermore, a little number of frameworks offers on-demand LiDAR data capabilities. Table 2 presents a comparison among these frameworks and our GVLiDAR proposal. The second column of the table indicates web applications that offer on-demand LiDAR data capabilities. In the other columns are indicated the following framework properties: the maximum amount of points allowed; provision of geospatial measuring tools over 3D point cloud; interactive rendering capabilities (considering the ability to display at least 10 million point cloud). Unlike the other solutions, the maximum amount of points GVLiDAR is capable of handling is only determined by the video memory (VRAM) available in the client GPU. Most current GPUs are able to store dozens or hundreds of million points which is a clear advantage over the reset of solutions being only *Potree* capable of handling larger datasets. We should point out here that the amount of points GVLiDAR is capable of handling in real time may be smaller than the point limit associated to the VRAM, so this, the performance delivered by the client GPU would be also decisive in order

Table 2. Comparative of LiDAR frameworks regarding web applications, the maximum number of points allowed, measuring tool available and performance in terms of real-time interaction

LiDAR framework	Web application data demand	Max. points (millions)	Measuring Tool	Real-time Interaction
IDECanarias	√	∼0.35		
LiDAR Online	√	∼1		
Lidarview		∼10		√
LAStools		No limit		
FugroViewer		No limit		√
Plas.io		16		√
Potree		No limit	√	√
GVLiDAR	√	Size of VRAM	√	√

to stablish the point limit of GVLiDAR. These issues will be further discussed in Section 4.

3. GVLiDAR

GVLiDAR is a web framework specially designed to accomplish detailed analysis and complex measurements over LiDAR 3D point clouds. It allows users to retrieve data on demand from queries based on spatial restrictions which makes possible to visualize and process a chosen region without transferring unneeded points, and regardless of the number and distribution of the source files acquired during different flights, allowing, for instance, the visualization of a road stored over multiple files. Here we should stress that experts consulted (in the field of agronomy engineering) required the performing geospatial measurements directly over the 3D point cloud and the visualization of all LIDAR data from a point of view. In technical disciples, not only a realistic representation of the data is necessary, but also the capacity to perform fast and accurate measures over the objects and terrains under study. Therefore, our proposal is not out-of-core, with a view to include out-of-core techniques in the following versions as an option. Figure 1 shows the general system environment. This framework can be considered as client-server architecture, having two clearly separated parts. The first part is a conventional web server, like Apache HTTP Server, which stores the whole system and the LiDAR files. The second part includes any client-side WebGL compatible web browser. This allows multiple users to access the system through their browsers, rendering the final image in their own platform.

The framework uses three different programming languages and the WebGL graphics API. HTML and JavaScript, both executed by the CPU, are responsible for tasks such as handling I/O events, generating the user interface, requesting and loading LiDAR data from our servers and performing the geospatial measurements. Although WebGL is also JavaScript code, it is a very distinctive part of the framework. It is the intermediary between the CPU and GPU, allowing communication and information delivery to the GPU. Finally, OpenGL Shading Language (GLSL) (The Khronos Group Inc. 2015a; Munshi, Ginsburg, and Shreiner 2008) is used to program the GPU shaders, *Vertex Shader* and *Fragment Shader*. *Shaders* are programs that provide a significant flexibility in order to implement rendering

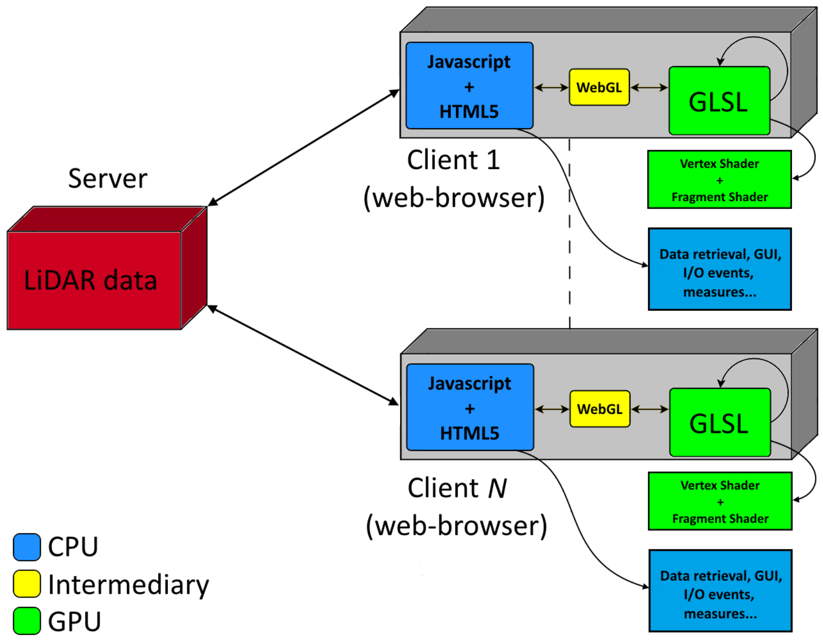

Server
LiDAR data
Javascript + HTML5
WebGL
GLSL
Client 1 (web-browser)
Vertex Shader + Fragment Shader
Data retrieval, GUI, I/O events, measures...
Javascript + HTML5
WebGL
GLSL
Client N (web-browser)
Vertex Shader + Fragment Shader
Data retrieval, GUI, I/O events, measures...
CPU
Intermediary
GPU

Figure 1. General structure of GVLiDAR.

and computational algorithms on GPUs. In GVLiDAR shaders are used for tasks such as point color generation, point erasing or mouse picking, which enables all the geospatial measurements implementation.

From the perspective of the users, the framework consists of two separate web pages. The *Selection* page (see Figure 2) shows a map through GoogleMaps API 3.0. Over this map users are able to obtain their geographic location or select a region to work with. Once users have made a selection, the *Visualization* page (see Figure 3) pops up showing a rendering of the point cloud available for the selected region. In this page, different LiDAR properties such as height, intensity, classification or return number, can be chosen in order to change the way the scene is rendered. There are also different camera options marker tools and render options such as point size, line width or projection mode.

One of the main differences between GVLiDAR and other LiDAR frameworks is the incorporation of geospatial measurements that can be performed directly over the point clouds. GVLiDAR makes it possible to measure distances between points, measure the height of structures, generate projected areas, triangulate points sets, create building facades or remove points. For example, Figure 6 shows the LiDAR data from Riazor Stadium (A Coruña, Spain) where a height measurement is being taken, from the top of a tower, located at one side of the stadium, to the grass field. Figure 7 shows another example of measurement, in this case a projected area (two-dimensional area measurement of a three-dimensional object by projecting its shape on to an arbitrary plane) is created covering different parts of the surface of the river Oitaven (Pontevedra, Spain).

The measurement tools available on other frameworks are very basic and not focused on the needs of professionals that use LiDAR point clouds, as it is the case of the agroforestry experts we are working with. Common and simple actions such as get geospatial information about the points on the image or even measure distances cannot be directly done in most of the applications commented in the previous section. A very similar problem happens measuring heights of structures, in some cases they must be manually calculated by the user from the angles given

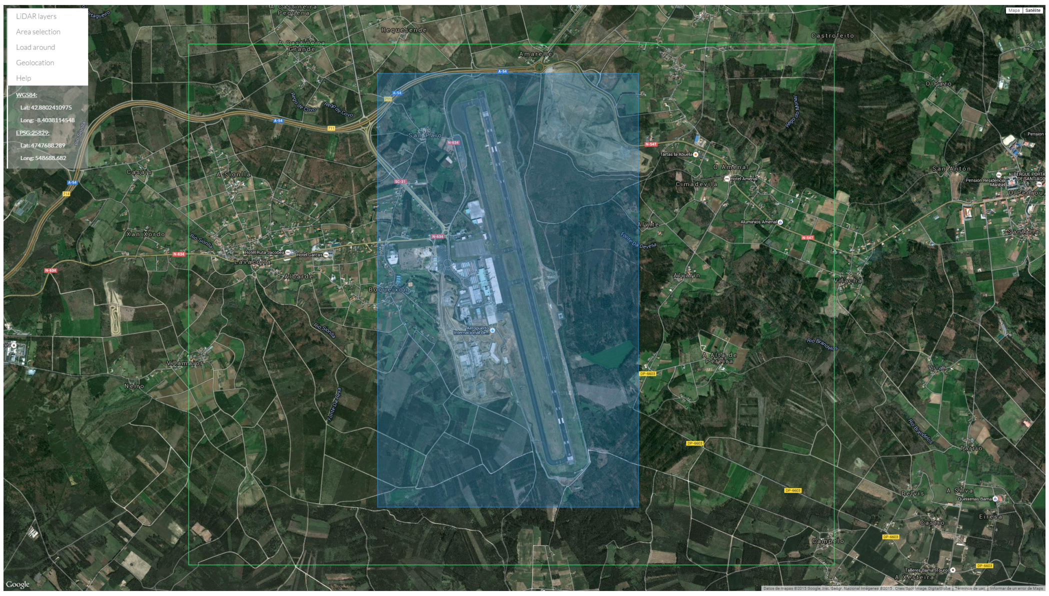


Figure 2. *Selection* page of GVLiDAR over Santiago de Compostela (Spain) International Airport. In this page we use the GoogleMaps API in order to help users to locate their areas of interest. The green rectangle represents the area of the map that contains point data while the blue rectangle represents the specific data selection done by the user.

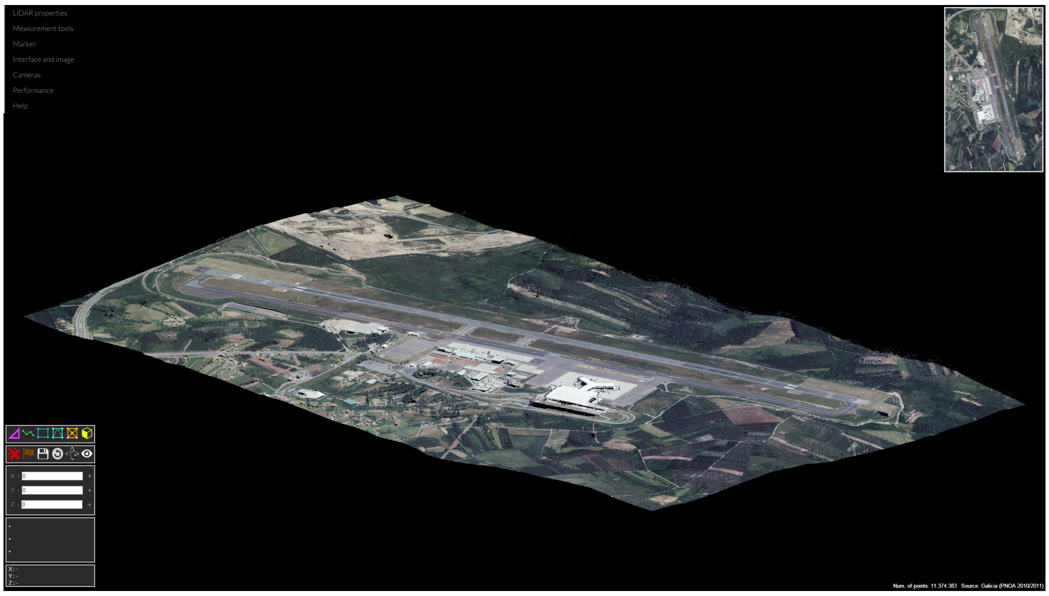


Figure 3. *Visualization* page of GVLiDAR rendering LiDAR data from Santiago de Compostela (Spain) International Airport. The point cloud shown matches with the selection done by the user in *Selection* page.

by angle tools or even they cannot be done in some situations. Most of the measures provided by the applications demand to pick up rendered points in the image, so that, a height measure like the example of Figure 4 is not possible. The green dot is generated in GVLiDAR from the pink and the blue point, so that, the green vertical line measures the distance from the top of the mountain down to its bottom. This type of user dots (like the green one) cannot be created in most of the current frameworks. Furthermore, areas they allow to measure are only projected horizontally while in GVLiDAR the areas can be projected on any plane allowing to measure facades or slopes. Also, some of them do not allow to triangulate points

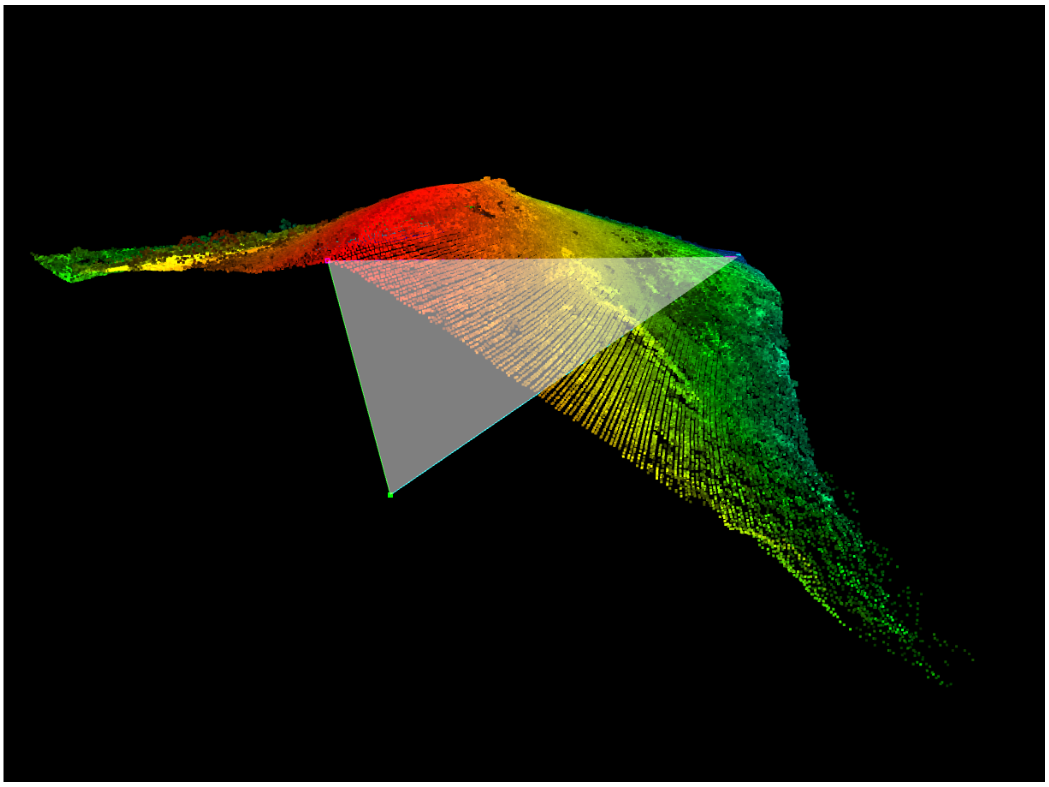


Figure 4. Point cloud render of a mountain from the PNOA dataset. The figure shows a vertical height measure (white triangle) of a mountain from its top to a plane placed at the height of a near river. The upper vertex of the triangle is placed on the top of the mountain, the lower right vertex determines the horizontal plane of the measure and is placed at the river height. Both vertex are user defined. Finally, the union of the upper vertex and the lower left vertex defines the vertical height.

in order to accurately measure complex surfaces and its volumes are just cubes or they are capable of doing a triangulation but they do not provide any type of measurement. To sum up, the measures available on other frameworks are far from being appropriate for many professionals in fields such as agronomy, urban planning or civil engineering.

Another feature presented in GVLiDAR is the possibility of overlapping the point cloud over a Digital Terrain Model (DTM). This, for instance, allows researchers who develop algorithms which generate DTMs from point clouds to compare their algorithm results (the DTMs) with their original data (the point clouds). Furthermore, it allows to judge the quality of any DTM overlapping it with a high dense and precise point cloud. These type of visual analysis demands to render as much points as possible in order to obtain good results. Inaccurate point clouds derived from hard under sampling or point interpolations may not have enough quality for these type of analysis or other accurate measures. Figure 5 shows a DTM overlapping a LiDAR point cloud obtained from different sources, this test allows to compare the quality of the first one.

3.1. *Implementation strategies*

In this section, we shall now go on to study a number of details in the design and implementation of GVLiDAR in order to obtain flexibility and low storage requirements.

In practice, some parts of the information stored in LAS files are often not useful

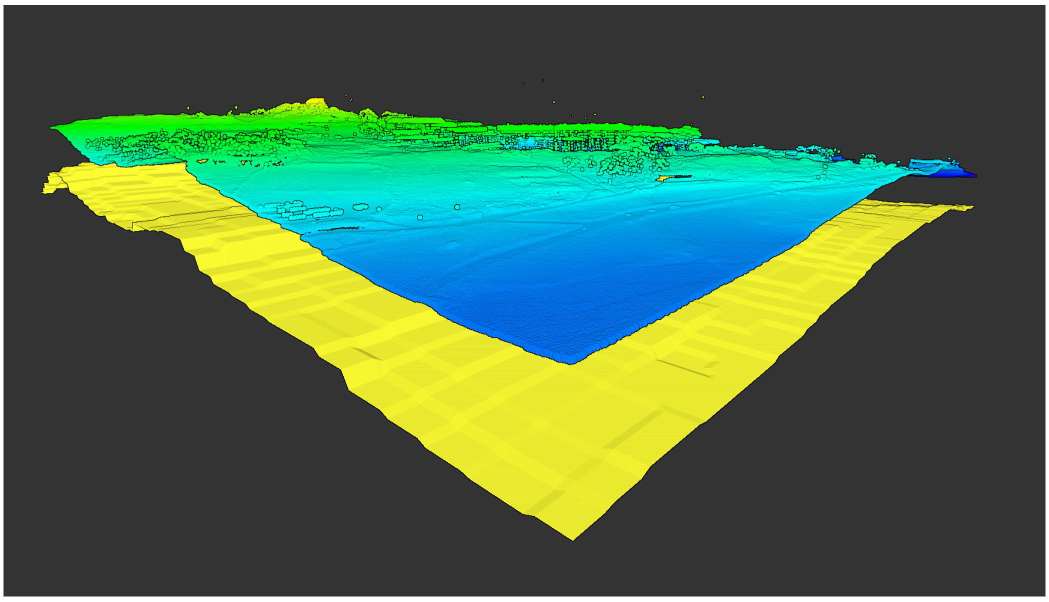
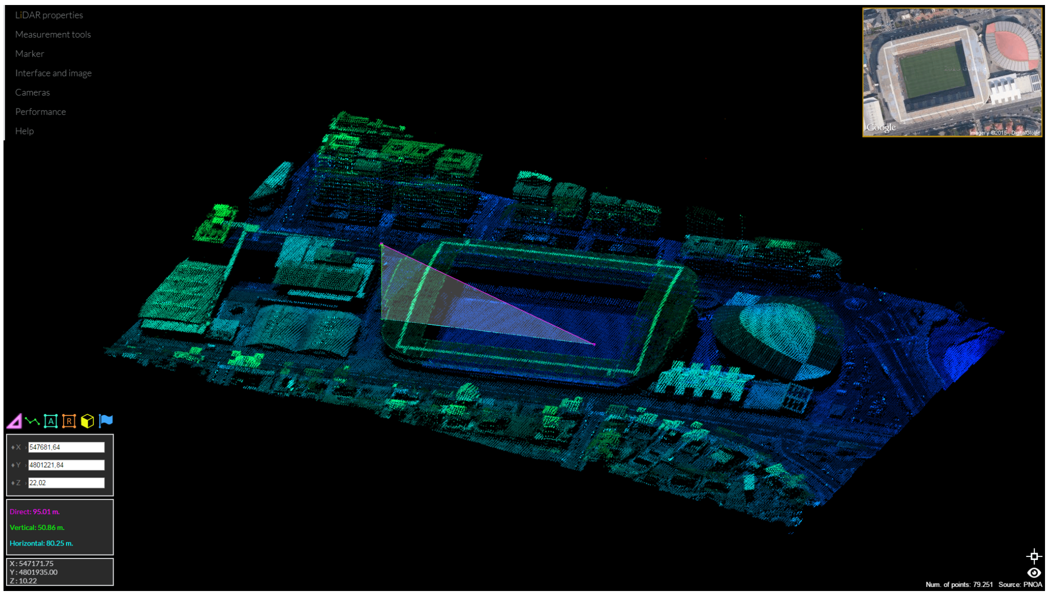


Figure 5. DTM overlapping a LiDAR point cloud in order to compare the quality of the first one.



X 547681,64
Y 4801221,84
Z 22,02

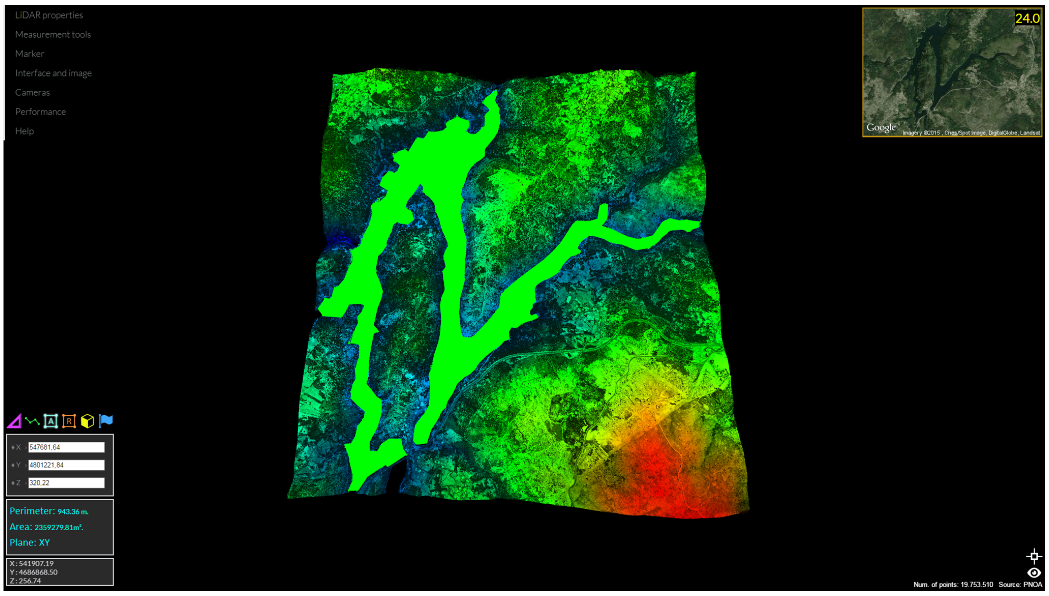
Direct: 95.01 m.
Vertical: 50.86 m.
Horizontal: 80.25 m.

X : 547171.75
Y : 4801935.00
Z : 10.22

Num. of points: 79.251 Source: PNOA

Figure 6. Point cloud render of the Riazor stadium (A Coruña, Spain). The figure shows the vertical height measure (white triangle) of a tower next to the stadium. Upper vertex is placed on the top of the tower, lower right vertex determines the horizontal plane of the measure. Both vertex are user defined. Finally, the union of the upper vertex and the lower left vertex defines the vertical height.

from the point of view of visualization or the geospatial measurements which unnecessarily increases the size of these files, demanding longer download times and storing capacity. Owing to this, and because nowadays point clouds may easily contain many millions of points, it is essential to apply a compression of the original LiDAR files in order to reduce the amount of data to be transferred and stored. For example, the complete LiDAR data of Galicia (a region inside Spain) gathered by the National Program of Aerial Ortophography (Plan Nacional de Ortofotografía Aérea, PNOA (Instituto Geográfico Nacional 2015)) has about 900 GB of information; nonetheless, after removing all the "useless" data, its size is reduced to 450 GB, a reduction of 50%. The superfluous information in these files includes:



X 547681,64
Y 4801221,84
Z 320,22

Perimeter: 943.36 m.
Area: 2359279.81m².
Plane: XY

X : 541907.19
Y : 4686868.50
Z : 256.74

Num. of points: 19 753 510 Source: PNOA

Figure 7. Point cloud render of the Oitaven river and surroundings (Pontevedra, Spain). The bright green area represents a projected area measurement covering the entire river surface.

the entire header block, all VLRs, all EVLRs and point properties such as *Scan Direction Flag*, *Edge of Flight Line*, *Scan Angle Rank*, *User Data* and *Point Source ID*. Furthermore, some LAS files may contain fields with all their values set to *null* or zero, so they become also useless data. PNOA files has a *RGB* property in each point record but it is always set to zero, which represents 12 bytes of worthless information per point, so that, this property is also deleted.

In addition to the problems described above, the large sizes of the original LiDAR files make hard to implement an efficient free terrain selection directed by the user. It is often desirable to select a small area to work with but it may involve downloading a large file with many more points than necessary. Moreover, the selection of a tiny zone in the intersection of 4 large files entails downloading all 4 files and then discarding most of the data obtained. To solve this problem all LiDAR files were subdivided into smaller files of no more than 5 MB, significantly reducing the amount of data requested. The limit of 5 MB allows the optimal utilization of web cache of the most commonly used browsers. A web cache system stores web documents, so that, subsequent requests may be satisfied from the cache. Note that files used by GVLiDAR are variable in size. During the pre-processing stage the files created are ensured to have a size less than 5 MB but they do not have a fixed size, their size may vary from a few KBs to almost 5 MBs depending on the amount of points stored. Note here that the pre-processing is not done each time a user retrieves data from the server. All the datasets are pre-processed offline once and the results are stored in our server, after that all files are served like regular files with no extra computational requirements. Any file-server such as Apache HTTP Server or Express (NodeJS) are capable of attend a large number of users retrieving files.

The small size of the files along with a data distribution in the form of a grid tile hierarchy allows GVLiDAR to perform efficient data queries based on spatial restrictions. Figure 8 shows a very simple example of the process. The yellow circles are user-defined points which delimit the region of interest (ROI). This is done in the Selection Page described on Section 3 of the paper. The green square represents the area covered by a LAS file. The red squares are the files created during our pre-

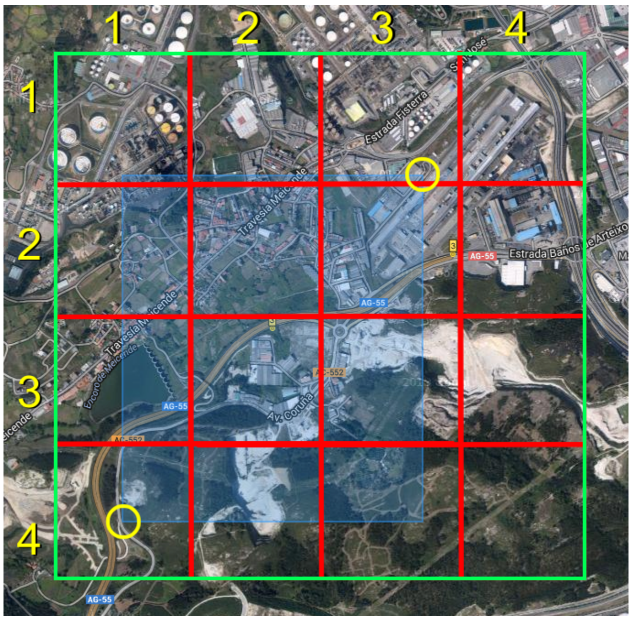


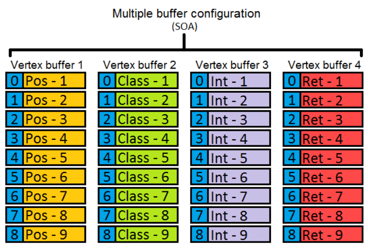
Figure 8. Simple schema representing a grid tiles hierarchy (Numbers along with green and red squares) with a data query over it (blue square and yellow circles).

process stage from the LAS file. The area covered is the same for all pre-processed files ensuring the file sizes are all below 5 MB in order to not disable browser cache. Each file is identified by its grid position. So this, knowing the underlining grid tiles hierarchy GVLiDAR is able to quickly obtain the list of files that must be retrieved from server based on the user-defined ROI. Following the example of the Figure 8, GVLiDAR first takes the two geographical points defined by user (inside the yellow circles in the figure) and calculates the files needed for obtaining the data marked on the blue rectangle (the ROI). Finally GVLiDAR retrieves from server the previously obtained file list. Note that the fourth column is not retrieved, saving time and memory. The points outside the ROI are fast discarded in client side during the load process. As commented in Section 4.3, these discarded points act as pre-cached data that will be immediately accessed when demanding an adjacent ROI. The extra storage and retrieval time used by the discarded points is a reasonable cost due to de small size of the files and the pre-cache factor.

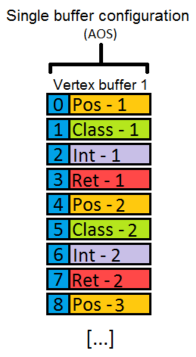
In our approach, additional information, such as GPSTime or RGB, could be easily included in auxiliary files if they were required in the future. These files would be stored apart from the current data and only retrieved from server if required. This provides flexibility and scalability allowing to include new properties as needed and even including custom properties not included in LAS format. Currently, the LiDAR properties stored in our structure are the most commonly used and valuable according with the group of experts in the field of agroforestry engineering we work with. This way, if new information would be included, the auxiliary files would not be sent to the majority of the clients that just want to visualize the point clouds using basic properties, they would be retrieved from server just when a specific client requires it.

3.2. *High performance visualization strategies*

In this section, we shall now go on to describe strategies in order to achieve high performance visualization.



(a)



(b)

Figure 9. The two most common ways of storing vertex attributes: 9(a) Structure of arrays (SOA). 9(b) Array of structures (AOS).

The two most common ways of storing vertex, or point attributes, are the array of structures (AOS) and the structure of arrays (SOA). Figures 9 shows the main differences between the two types. The AOS (Figure 9(b)) is implemented by using a single buffer configuration, storing all point attributes together. On the other hand, the SOA (Figure 9(a)) is implemented by using a multiple buffer configuration; each buffer stores one type of attribute. In original LAS files, all properties are stored *per point*, using an AOS, while for the design of GVLiDAR the SOA was chosen to program the GPU shaders in order to obtain a better performance. The fact that original LAS files and inputs of the *Vertex Shaders* use different kinds of structures demands a structure transformation from AOS to SOA. Performing this transformation in the client would be inefficient; hence, it is done in the server during a pre-processing stage.

As discussed above, that LAS file format sequentially stores all properties relative to a specific point. From the point of view of visualization, only its coordinates (x, y, z), classification values, return and intensity are used. To manipulate binary files in JavaScript it is necessary to use the *slice* function and data type transformations from binary to *byte*, *short*, *integer* or *float*. Function *slice* allows data in a binary file to be retrieved from a starting byte to a final byte. Since LAS files have a specific and defined structure (see Section 2) it is easy to ascertain accurately the address where the required information can be found. Thus, this distribution implies the need to perform, for each point, four calls to the function *Slice*. In the first call, point coordinates are extracted, in the second call, intensity, and finally the return and classification. As reference, certain LAS files from PNOA in which each file contains close to 3,000,000 points, around 12 million calls to *Slice* and 12 million data transformations are needed. These operations heavily penalize data load times. The optimization strategy is based on the representation of point clouds using an array of structures.

In the GVLiDAR proposal, the representation is computed in a compact and efficient way as a pre-processing stage in the server. This information is sent to the GPU client, where the point attributes corresponding to a specific region are rendered. This strategy makes it possible to reduce the number of *slice* functions to 4, one for each type of information (coordinates, classification, return and intensity). Table 3 shows the size of the new structure with N points.

The group of agroforestry experts we work with was not interested in the numeric values of the return properties, instead, they demanded information in the form of return tags, that is, a classification of each point based on its return information

Table 3. Data distribution into the pre-processed LAS files used by GVLiDAR.

Item	Size (Bytes)
X, Y, Z	$3 \times 4 \times N$
Intensity	$4 \times N$
Ret. + Class.	N

mixing the return number of the point and the number of returns of its pulse. So this, we have created 5 tags or categories that can be stored in 3 bits instead of the previously 5 bits used in the ASPRS specification (see Section 2):

(1) First return (return number 1 out of N pulses, being $N > 1$). This tag corresponds to objects like forest canopy.
(2) Middle return (return number n out of N pulses, being $N > 2$, $n \neq 1$ and $n \neq N$). This tag represents points placed between the canopy and the ground, like branches or leaves.
(3) Last return (return number N out of N pulses, being $N > 1$). A last return represents points placed in the ground.
(4) No return (return number 0 out of 0 pulses). This tag denotes an artificial point. These are points added to the point cloud after being collected by the laser scanner. Normally used to outline rivers or lakes.
(5) Unique return (return number 1 out of 1 pulse). This tag correspond to points obtained from a solid surface such us buildings, roads or stones.

These 5 categories allow GVLiDAR to filter the point cloud in many useful ways combining the different categories.

The coordinates stored in the original LAS files are not the actual coordinates of each point; each coordinate has to be adjusted according to a scale and offset. During the pre-processing stage the coordinate values in the files are analysed, if they can be stored as 32 bits float numbers without losing precision they are stored this way. This prevents to do extra operations in the client side each time the point clouds are loaded. If the use of floating point numbers implies losing precision offsets and scales are used. Each dataset owns a metadata file (JSON format) containing all information required to retrieve, load and render the dataset properly. These metadata files store information such as the name of the dataset, its coordinate system and the offset and scale (if used) of the point clouds. So this, the structure generated in our proposal stores the actual values when possible, avoiding many operations that should be performed during the load stage.

4. Results and Application

In this section, the results of the evaluation of GVLiDAR are presented and analysed. Our test platform is an Intel Core i7 4790, 32 GB of RAM DDR3 with a Nvidia GeForce GTX Titan. The screen resolution was 2048×1152 under different browsers.

LiDAR data used come from two repositories: *PNOA* and *CA13*. The datasets were obtained by airborne sensors. The dataset for the *PNOA* test is available in the Spanish GIS database (IDEE) (Instituto Geográfico Nacional 2015). Specifically, we have used the region of Galicia that contains over 27.6 billion of points (see Figure 10). The airborne LiDAR survey of the study area was performed and processed in 2009 and 2010 by PNOA (Plan Nacional de Ortofotografía Aérea) with a LiDAR

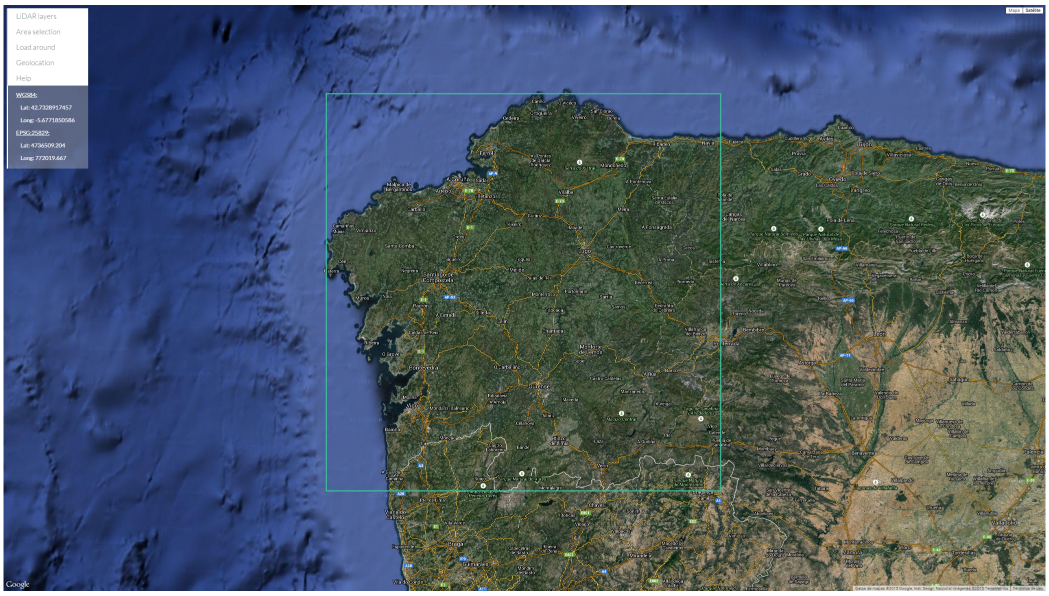


Figure 10. *Selection* page of GVLiDAR. We employ GoogleMaps API in order to show users the datasets available and their extension. The green square delimits the area with point data, in this case, the *PNOA* (Galicia, Spain) dataset used in this paper.

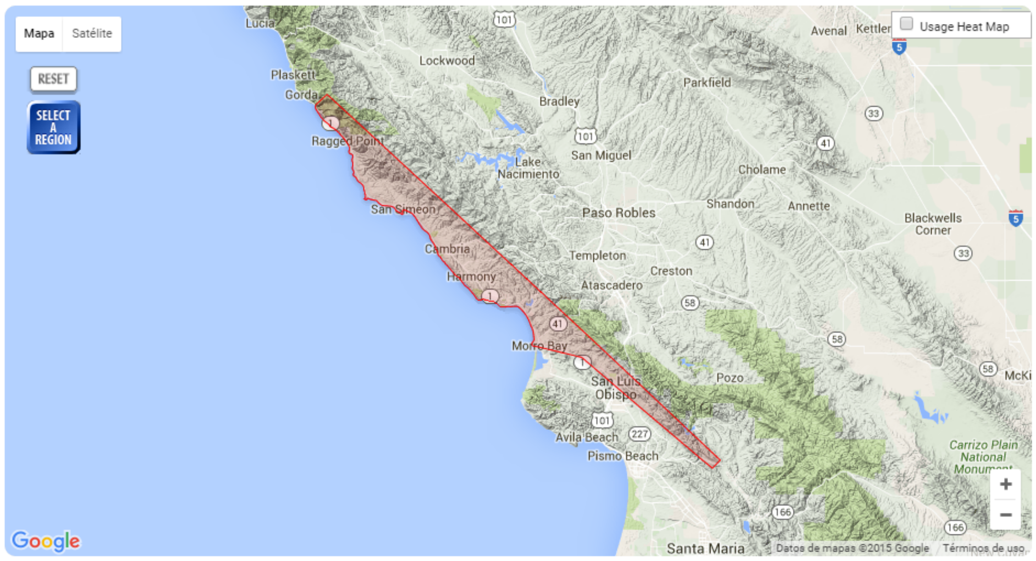


Figure 11. OpenTopography website, specifically its data selection tool. The red zone indicates *CA13* dataset that is located in San Luis Obispo County, California.

data acquisition of 0.5 $point/m^2$. *CA13* (OpenTopography 2015) (see Figure 11) is located in San Luis Obispo County, California, and extends approximately 75 miles north to Monterey County. Overall LiDAR and orthophotography acquisition of the DCCP San Simeon survey area occurred between January 29 and February 25, 2013, and encompasses approximately 198,000 acres (801 square kilometers). For optimal capture of the intertidal zone, WSI acquired LiDAR data of the coastline during seasonal low tides between, February 7 and February 10, 2013. The point density of the image is 22.06 $points/m^2$ and dataset has 17.7 billion points.

The next subsections focus on the analysis of GVLiDAR in three different key points: the analysis of the functionality, the performance in terms of FPS and data retrieval time.

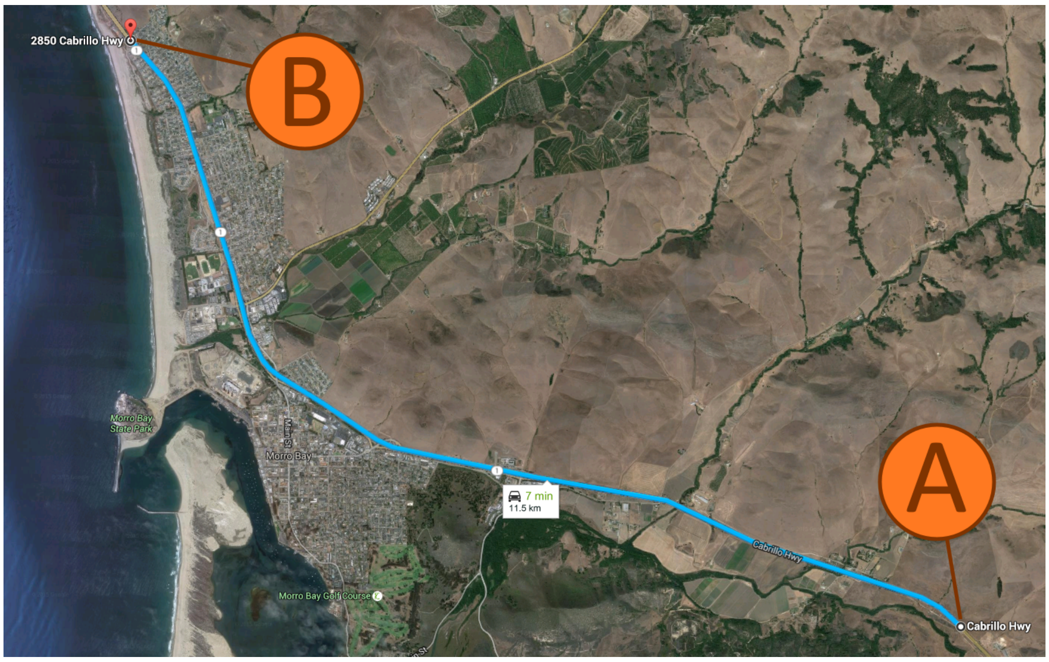


Figure 12. Distance measurement of one of the roads contained in the *CA*13 LiDAR dataset, from *A* to *B*, using GoogleMaps only.

4.1. *Functionality*

In this section, experiments were performed on the system described above and the browser Windows Chrome 42.0.2311.135 (64 bits). As previously mentioned, few LiDAR web frameworks present a measurement tool. So the functionality of GVLiDAR is analysed in comparison with *Potree* proposal using *CA13* dataset. The measurement parameters of the analysis depend on the subject that tool is applied to. We have recreated a framework that must take accurate measurements about the length of roads. For instance, this is necessary for institutions in order to calculate the cost of road maintenance. In general, the measurements require working with a point of view close to the roads in order to obtain a high LOD of them. Figure 12 shows the zone of road that we want to measure, starting on position *A* and finishing on position *B*.

Potree visualizes up to 1 million points using a multi-resolution technique in order to keep good frame rates and a low memory consumption. This limit can be raised up to 4 million; nevertheless, the default configuration of the framework was used due to 1 million points are enough to show LiDAR data with high detail in order to measure the road. *Potree* uses about 1.3 GB of RAM in the beginning of the procedure of measurement on position *A*; however, it needs up to 4GB of RAM when the user reaches position *B*. All points rendered during the procedure of measurement are kept in memory browser which could reach the limitations of software or hardware due to the requirements of memory used. Chrome has a limit of 4GB of RAM per tab; therefore, *Potree* causes the closing of its tab during the test losing all the measures. This memory limitation can be found in all 32 bits web browsers and the 64 bits versions of Chrome for Windows. Here we should also stress that *Potree* allows the measurement of distances of a given section but, it does not give the total distance of a pathway.

On the other hand, GVLiDAR provides the capability of transferring only chosen sections of data. This enables to visualize and process the data without transferring unneeded points. Figure 13 shows the section of the road that the user loads in

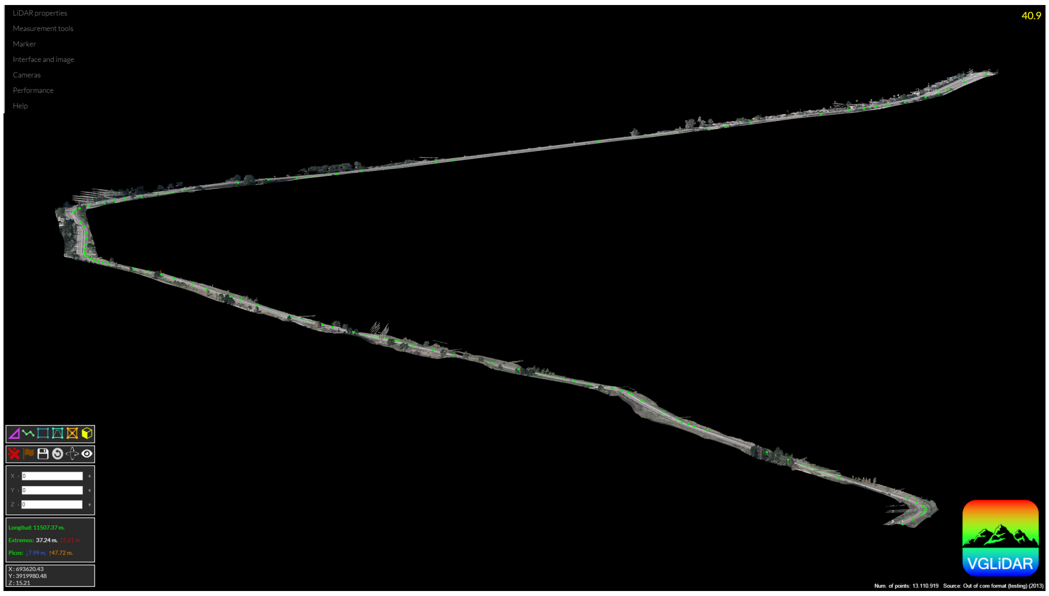


Figure 13. GVLiDAR point cloud render of the road from $CA13$ with the same distance measurement realized previously in GoogleMaps.

GVLiDAR to measure the road from position A to position B in $CA13$ dataset. About 500 MB of RAM and 800 of VRAM are the storage requirements in this case. GVLiDAR has a considerable margin regarding the 4GB Chrome browser limit in order to measure the length of the road while *Potree* may experience stability problems due to the memory management.

In summary, with high levels of resolution and an intense use of the application, *Potree* could fault due to the high requirements of storage. However, GVLiDAR exploits the capability of on-demand data that allows minimizing the storage requirements and generates the measurements on-the-fly without the need of further pre-processing.

4.2. *Performance*

Performance in terms of FPS is another important aspect to be analysed. In this section, experiments were performed on the system described above and the browser Firefox 41 (64 bits). The point clouds used in our test are shown in Figure 14, indicated with white squares (see Figure 14(a)). Figure 14(b) shows a zoom of the largest point cloud inside one of the zones.

Figure 15 shows the results of the performance tests for each dataset where N stands for the number of points rendered in the *Visualization* page. $VP1$ and $VP2$ are two different points of view in which the camera is located in the 1/4 and 3/4, respectively, of the full 2D convex hull. This graph shows the good results in terms of frame rate obtained by GVLiDAR for five test regions. Thus, for example, when 20,179,576 points are rendered, up to 60 FPS are achieved, while maintaining rendering quality. For larger regions (up to 51 million points) GVLiDAR achieves real-time interaction (above 20 FPS) for both VP1 and VP2. Even rendering 103 million points it is capable of reaching 42 FPS for VP2. Observe that GVLiDAR is able to render up to 281 million points, which is not achieved in other applications with the exception of *Potree*.

Frameworks such as *IDECanarias* or *Lidar-Online* achieve similar performances for a small number of points rendered (350,000 or 1,000,000). Other frameworks

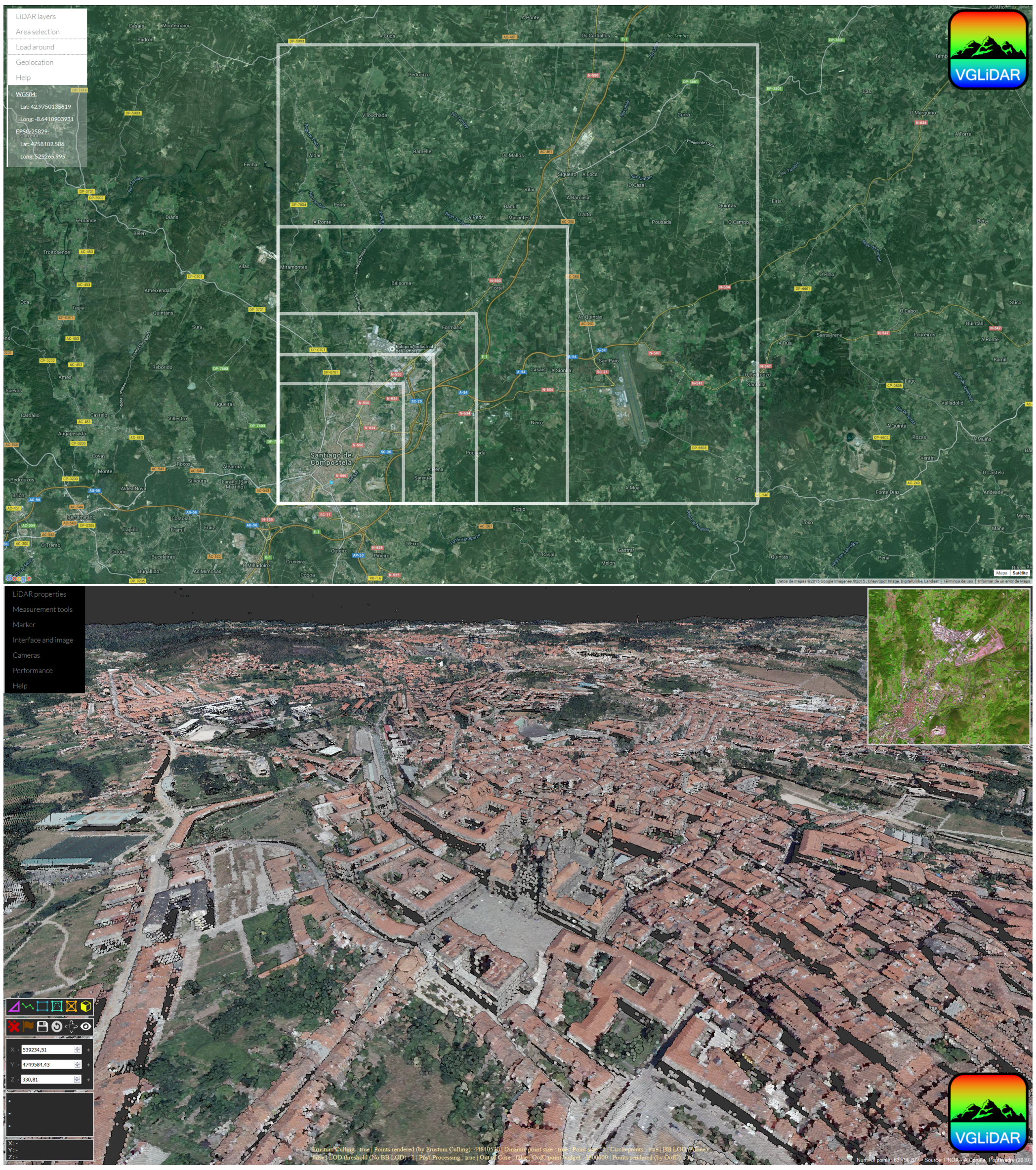


Figure 14. Test zones: (Top) Each white square indicates a selected test zone; (Bottom) Point cloud in the *Visualization* page for the largest zone, N = 281,152,780

such as *FugroViewer* and *Potree* achieve good results applying LOD techniques that remove points from the scene.

These performance tests have led us to stablish two different theoretical point limits in GVLiDAR considering the current GPU market. The maximum amount of points that can be stored in the 12 GB of VRAM available on the Titan X was 281 million, meanwhile, 103 million points is the performance limit of the GPU, that is, the maximum amount of points that is able to handle with relative good performance (between 10 and 42 FPS).

4.3. *Data retrieval and data load times*

In this section, retrieval times and load times were measured to confirm the practical usability of our framework. Specifically, we have considered the load times as the time spent during the client-side processing of the points (discard points, set

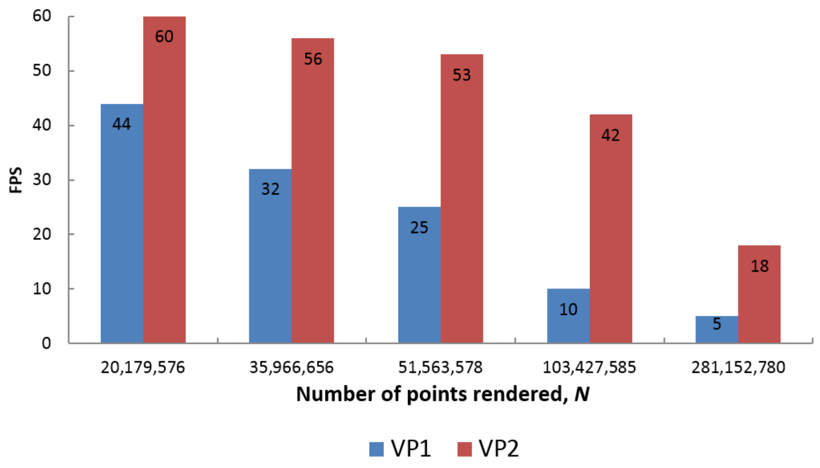


Figure 15. GVLiDAR Performance in terms of FPS, where N represents the number of points rendered in the *Visualization* page.

indexes, prepare data to send it to GPU buffers, among other tasks). The retrieval times are considered as the time required for moving the data from the server to the client machine. Therefore, the sum of both times (retrieval + load), which we will call *wait time*, represents the elapsed time between a users makes a data query and the point cloud is rendered on the screen. This experiment was performed on an Apache 2.215 server under CentOS that consists of an Intel Xeon ES-2603v3 with 64 GB of RAM, and a HP 4TB 6G SATA hard drive. The network transfer speed is 90 Mbps and the browser is Firefox 41 (64 bits).

Figure 16 shows how the load, the retrieval and the wait time vary with the number of points requested. The graph reflects the worst possible scenario where all data selected by the user must be entirely retrieved from server. It is quite clear from the graph that times increase almost linearly along with N due to the size of each point is always 17 bytes.

As commented during Section 3.1, browsers can store all retrieved data in the client local disk (under a given file size) in order to increase performance in subsequent retrievals. The data structure used by GVLiDAR allows this data caching. Therefore, considering the best possible scenario, where all selected data is already cached in the client disk, the retrieval time would be zero making the wait time equal to the load time. Furthermore, previously discarded points are cached; in consequence, the request of neighbouring areas can be accessed with lower load time.

Although the time spent retrieving non-cached data may become high when starting to retrieve more than 25 million points this results were obtained without using any kind of compression algorithms. Currently, we are working on a custom compressed version of our data structure that fits the needs of GVLiDAR and greatly reduces the retrieval times and the server disk requirements. First tests show that this new compression in conjunction with the GZIP algorithms applied by most servers (Apache, Express) would allow us to reduce the size of the LAS files by between 80% and 90% obtaining equal or even best results than using LASZip.

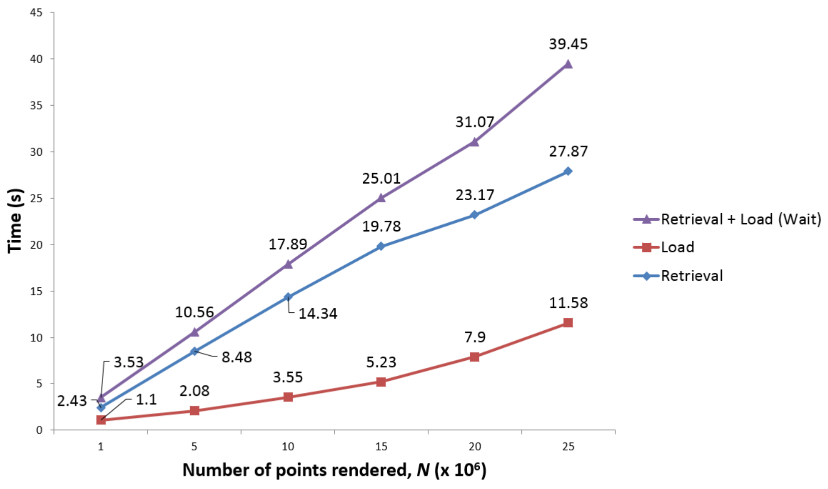


Figure 16. Retrieval times from remote server.

5. Conclusions

GVLiDAR is a web framework with a client-server architecture that allows to obtain LiDAR data on-demand. This enables to visualize and process the data without transferring unneeded points. Furthermore, while other web frameworks achieve good performance handling very limited point sets, GVLiDAR achieves real time interaction, being able to render much more data point sets. About 103 million points can be handled, and up to 281 million points could be rendered with an acceptable interaction. On the other hand, GVLiDAR enables geospatial measurements directly over the 3D point cloud such as area, length or volume. GVLiDAR outperforms other well-known and commonly used web or desktop LiDAR frameworks.

As future work, our primary focus is to achieve capabilities to work with larger zones using out-of-core techniques but maintaining a web framework aimed to retrieve data on demand. The objective of GVLiDAR is, in the near future, to offer as many public datasets as possible and allow users to easily upload their private point clouds, aiming to keep the advantages of web applications, instant-access and lack of setup or installation processes.

Acknowledgements

The LiDAR datasets used in this article belong to:

- LiDAR-PNOA data repository (Instituto Geográfico Nacional 2016). Provided by ©Instituto Geográfico Nacional de España.
- PG&E Diablo Canyon Power Plant (DCPP): San Simeon, CA Central Coast dataset (OpenTopography 2015). This material is based on LiDAR Point Cloud Data Distribution and Processing services provided by the OpenTopography Facility with support from the National Science Foundation under NSF Award Numbers 1226353 & 1225810.

Funding

This research has been supported by Galician Government (Xunta de Galicia) under the Consolidation Program of Competitive Reference Groups, co-founded by FEDER funds of the EU (Ref. GRC2013 / 055); under the Consolidation Program of Competitive Research Units, co-founded by FEDER funds of the EU (Ref. R2014/049); and by the Ministry of Economy and Competitiveness of Spain and FEDER funds of the EU (Project TIN2013-42148-P and TIN2016-75845-P).

References

Alexander Krivutsenko. 2015. “Lidarview.” Accessed: 10/2/2015. http://lidarview.com/.

American Society for Photogrammetry and Remote Sensing. 2011. “LAS specification.” Accessed: 10/2/2015. http://www.asprs.org/Committee-General/LASer-LAS-File-Format-Exchange-Activities.html.

Arrowsmith, J. R., and O. Zielke. 2009. “Tectonic geomorphology of the San Andreas Fault zone from high resolution topography: an example from the Cholame segment.” *Geomorphology* 113 (1–2): 70–81.

Bóo, M., M. Amor, and J. Döllner. 2007. “Unified hybrid terrain representation based on local convexifications.” *GeoInformatica* 11 (3): 331–357.

Boudreau, J., R.F. Nelson, H.A. Margolis, A. Beaudoin, L. Guindon, and D.S. Kimes. 2008. “Regional aboveground forest biomass using airborne and spaceborne LiDAR in Québec.” *Remote Sensing of Environment* 112: 3876–3890.

Brunori, C. A., R. Civico, F. R. Cinti, and G.Ventura. 2013. “Characterization of active fault scarps from LiDAR data: a case study from Central Apennines (Italy).” *International Journal of Geographical Information Science* 27 (7): 1405–1416.

Buján, S., E. González-Ferreiro, L. Barreiro-Fernández, I. Santé, E. Corbelle, and D. Miranda. 2013. “Classification of rural landscapes from low-density lidar data: is it theoretically possible?.” *International Journal of Remote Sensing* 34 (16): 5666–5689.

Dielmo 3D S.L. 2015. “Dielmo.” Accessed: 10/2/2015. http://www.dielmo.com/.

Fugro Geospatial Services. 2015. “FrugoViewer.” Accessed: 10/2/2015. http://www.fugroviewer.com/.

Gobierno de Canarias. 2015. “Visor IDECanarias.” Accessed: 10/2/2015. http://visor.grafcan.es/visorweb/.

González-Ferreiro, E.M., D. Miranda, L. Barreiro-Fernández, S. Buján, J. García-Gutiérrez, and U. Diéguez-Aranda. 2013. “Modelling stand biomass fractions in Galician Eucalyptus globulus plantations by use of different LiDAR pulse densities.” *Forest Systems* 22 (3): 510–525. http://revistas.inia.es/index.php/fs/article/view/3878.

Hasegawa, H., H. P. Sato, and J. Iwahashi. 2007. “Continuous caldera changes in Miyakejima volcano after 2001.” *Bulletin of Geospatial Information Authority of Japan* 54: 60–64.

Instituto Geográfico Nacional. 2015. “Plan Nacional de Ortofotografía Aérea (PNOA).” Accessed: 10/28/2015. http://pnoa.ign.es/presentacion.

Instituto Geográfico Nacional. 2016. “PNOA, download center.” Accessed: 04/11/2016. http://centrodedescargas.cnig.es/CentroDescargas/buscadorCatalogo.do?codFamilia=LIDAR.

Koca, C., and U. Güdükbay. 2014. “A hybrid representation for modeling, interactive editing, and real-time visualization of terrains with volumetric features.” *International Journal of Geographical Information Science* 28 (9): 1821–1847.

Kuder, M., and B. Zalik. 2011. “Web-Based LiDAR Visualization with Point-Based Rendering.” In *2011 Seventh International Conference on Signal-Image Technology and Internet-Based Systems (SITIS),* 38–45.

Landy, J. C., A. S. Komarov, and D. G. Barber. 2015. “Numerical and Experimental Evaluation of Terrestrial LiDAR for Parameterizing Centimeter-Scale Sea Ice Surface Roughness.” *IEEE Transactions on Geoscience and Remote Sensing* 53 (9): 4887–4898.

Lewis, P., C. P. Mc Elhinney, and T. McCarthy. 2012. "LiDAR Data Management Pipeline; from Spatial Database Population to Web-application Visualization." In *Proceedings of the 3rd International Conference on Computing for Geospatial Research and Applications,* Washington, D.C.. COM.Geo '12. 16:1–16:10. http://doi.acm.org/10.1145/2345316.2345336.

LiDAR Online. 2015. "LiDAR Online web tools." Accessed: 10/2/2015. http://www.lidar-online.com/tools/maps/.

Markus Schtz. 2015. "Potree." Accessed: 10/2/2015. http://potree.org/.

Moller, T.A., E. Haines, and N. Hoffman. 2008. *Real-Time Rendering, Third Edition.* A.K. Peters Ltd.

Munshi, A., D. Ginsburg, and D. Shreiner. 2008. *OpenGL ES 2.0 Programming Guide.* Addison-Wesley.

OpenTopography. 2015. "CA13, PG&E Diablo Canyon Power Plant (DCPP): San Simeon, CA Central Coast." Accessed: 10/28/2015.

Paredes, E.G., Montserrat Bóo, Margarita Amor, J.D. Bruguera, and J. Döllner. 2011. "Extended hybrid meshing algorithm for multiresolution terrain models." *International Journal of Geographical Information Science* 26 (5): 771–793.

Parisi, T. 2012. *WebGL - Up and Running.* O'Reilly Media.

Rapidlasso. 2015. "LAStools." Accessed: 10/2/2015. http://rapidlasso.com/lastools/.

Sallenger Jr., A. H., W. Krabill, J. Brock, R. Swift, M. Jansen, S. Manizade, B. Richmond, M. Hampton, and D. Eslinger. 1999. "Airborne laser study quantifies El Niño-induced coastal change." *Eos, Transactions, American Geophysical Union* 80 (8): 89–92.

Tarolli, P. 2014. "High-resolution topography for understanding Earth surface processes: Opportunities and challenges." *Geomorphology* 216: 295–312.

Tarolli, P., and D. G. Tarboton. 2006. "A new method for determination of Most Likely Initiation Points and the evaluation of Digital Terrain Model scale in terrain stability mapping." *Hydrology and Earth System Sciences Discussions* 3 (2): 395–425. https://hal.archives-ouvertes.fr/hal-00298671.

The Khronos Group Inc. 2015a. "OpenGL Shading Language." Accessed: 10/2/2015. https://www.opengl.org/documentation/glsl/.

The Khronos Group Inc. 2015b. "WebGL." Accessed: 10-2-2015. https://www.khronos.org/webgl.

Uday Verma. 2015. "Plas.io." Accessed: 10/2/2015. http://plas.io/.

Ventura, G., G. Vilardo, C.Terranova, and E. B. Sessa. 2011. "Tracking and evolution of complex active landslides by multi-temporal airborne LiDAR data: the Montaguto landslide (Southern Italy)." *Remote Sensing of Environment* 115 (12): 3237–3248.

Yan, W. Y., A. Shaker, and N. El-Ashmawy. 2015. "Urban land cover classification using airborne LiDAR data: A review." *Geomorphology* 158: 295–310.

Yang, B., W. Shi, and Q. Li. 2005. "An integrated TIN and grid method for constructing multi-resolution digital terrain models." *International Journal of Geographical Information Science* 19 (10): 1019–1038.

Zhang, J. 2010. "Multi-source remote sensing data fusion: status and trends." *International Journal of Image and Data Fusion* 1 (1): 5–24.